Peter M. D. Gray
Krishnarao G. Kulkarni
Norman W. Paton

Object-Oriented Databases

A Semantic Data Model Approach

C.A.R. HOARE SERIES EDITOR

OBJECT-ORIENTED DATABASES

A Semantic Data Model Approach

Peter M.D. Gray
University of Aberdeen, Scotland

Krishnarao G. Kulkarni
Digital Equipment Corporation, Colorado

Norman W. Paton
Heriot-Watt University, Scotland

Prentice Hall
New York London Toronto Sydney Tokyo Singapore

First published 1992 by
Prentice Hall International (UK) Ltd
Campus 400, Maylands Avenue
Hemel Hempstead
Hertfordshire, HP2 7EZ
A division of
Simon & Schuster International Group

Typeset in 10/12pt Times
by Mathematical Composition Setters Ltd,
Salisbury, Wiltshire

Printed and bound in Great Britain at
Dotesios Limited, Trowbridge

Library of Congress Cataloging-in-Publication Data

Gray Peter M.D., 1940-
Object-oriented databases: a semantic data model approach/ Peter M.D. Gray, Krishnarao G. Kulkarni, Norman W. Paton.
p. cm. —— (Prentice-Hall International Series in Computer Science)
Includes bibliographical references and index.
ISBN 0–13–630203–3
1. Object-oriented databases. I. Kulkarni, Krishnarao G. II. Paton, Norman W. III. Title. IV. Series.
QA76.9.D3G7222 1992
005.75--dc20 91–32213
CIP

British Library Cataloguing in Publication Data

Gray, Peter M.
Object-oriented databases: A semantic data model approach.
I. Title II. Kulkarni, Krishnarao
III. Paton, Norman W.
005.74

ISBN 0–13–630203–3

1 2 3 4 5 96 95 94 93 92

Object-Oriented Databases

Prentice Hall International Series in Computer Science

C. A. R. Hoare, Series Editor

BACKHOUSE, R. C. *Program Construction and Verification*
DEBAKKER, J. W., *Mathematical Theory of Program Correctness*
BARR, M. and WELLS, C., *Category Theory for Computing Science*
BEN-ARI, M., *Principles of Concurrent and Distributed Programming*
BIRD, R. and WADLER, P. *Introduction to Functional Programming*
BORNAT, R., *Programming from First Principles*
BUSTARD, D., ELDER, J. and WELSH, J., *Concurrent Program Structures*
CLARK, K. L. and McCABE, F. G., *Micro-Prolog: Programming in logic*
CROOKES, D., *Introduction to Programming in Prolog*
DAHL, O-J., *Verifiable Programming*
DROMEY, R. G., *How to Solve it by Computer*
DUNCAN, E., *Microprocessor Programming and Software Development*
ELDER, J., *Construction of Data Processing Software*
ELLIOTT, R. J. and HOARE, C. A. R., (eds), *Scientific Applications of Multiprocessors*
GOLDSCHLAGER, L. and LISTER, A., *Computer Science: A modern introduction (2nd edn).*
GORDON, M. J. C., *Programming Language Theory and its Implementation*
GRAY, P. M. D., KULKARNI, K. G. and PATON, N. W. *Object-Oriented Databases*
HAYES, I., (ed.), *Specification Case Studies*
HEHNER, E. C. R., *The Logic of Programming*
HENDERSON, P., *Functional Programming: Application and implementation*
HOARE, C. A. R., *Communicating Sequential Processes*
HOARE, C. A. R. and JONES, C. B. (eds), *Essays in Computing Science*
HOARE, C. A. R. and SHEPHERDSON, J. C. (eds), *Mathematical Logic and Programming Languages*
HUGHES, J. G., *Database Technology: A software engineering approach*
HUGHES, J. G., *Object-oriented Databases*
INMOS LTD, *Occam 2 Reference Manual*
JACKSON, M. A., *System Development*
JOHNSTON, H., *Learning to Program*
JONES, C. B., *Systematic Software Development using VDM (2nd edn)*
JONES, C. B. and SHAW, R. C. F. (eds), *Case Studies in Systematic Software Development*
JONES, G., *Programming in occam*
JONES, G. and GOLDSMITH, M., *Programming in occam 2*
JOSEPH, M., PRASAD, V. R. and NATARAJAN, N. A., *A Multiprocessor Operating System*
KALDEWAIJ, A., *Programming: The Derivation of Algorithms*
KING, P. J. B., *Computer and Communication Systems Performance Modelling*
LEW, A., *Computer Science: A mathematical introduction*
MARTIN, J. J., *Data Types and Data Structures*
McCABE, F. G., *High-Level Programmer's Guide to the 68000*
MEYER, B., *Introduction to the Theory of Programming Languages*
MEYER, B., *Object-oriented Software Construction*
MILNER, R., *Communication and Concurrency*
MORGAN, C., *Programming from Specifications*
PEYTON JONES, S. L., *The Implementation of Functional Programming Languages*
PEYTON JONES. S. L. and LESTER, D., *Implementing Functional Languages*
POMBERGER, G., *Software Engineering and Modula-2*
POTTER, B., SINCLAIR, J., TILL, D., *An Introduction to Formal Specification and Z*
REYNOLDS, J. C., *The Craft of Programming*
RYDEHEARD, D. E. and BURSTALL, R. M., *Computational Cetegory Theory*
SLOMAN, M. and KRAMER, J., *Distributed Systems and Computer Networks*
SPIVEY, J. M., *The Z Notation: A reference manual*
TENNENT, R. D., *Principles of Programming Languages*
TENNENT, R. D., *Semantics of Programming Languages*
WATT, D. A., *Programming Language Concepts and Paradigms*
WATT, D. A., WICHMANN, B. A. and FINDLAY, W., *ADA: Language and methodology*
WELSH, J. and ELDER, J. Introduction to Modula 2
WELSH, J. and ELDER, J. Introduction to Pascal (3rd edn)
WELSH, J., ELDER, J. and BUSTARD, D., *Sequential Program Structures*
WELSH, J. and HAY, A., *A Model Implementation of Standard Pascal*
WELSH, J. and McKEAG, M., *Structured System Programming*
WIKSTRÖM, Å., *Functional Progrmaming using Standard ML*

Introduction

This book concerns an interesting new development in computing science: the storage of networks of objects and procedures in a database, potentially for re-use by other users. This new development has been referred to as *data-intensive programming in the large* (Zdonik and Maier, 1990), and it combines ideas from two different fields. It takes ideas from object-oriented programming concerning the creation and manipulation of objects, and it takes ideas from database research concerning the sharing and long-term storage of large numbers of objects. These are two parts of computing science that have not had much in common, and in some ways it is like the problems the physicists face near a black hole, where they have to take account of both quantum theory and relativity – two very different theories which evolved independently!

The consequence of this is that we have a new field with very little agreed terminology, and practitioners using different words for almost identical concepts! A further problem is that different people emphasize object-oriented programming features or database features, according to their background. In fact, any real object-oriented database (OODB) must involve a skillful blend of ideas from both. In this book we shall stress ideas from a particular branch of database theory called semantic data modeling. We do this because of the powerful and intuitive semantic constructs this provides, which we feel are appropriate for describing objects. We shall concentrate on a particular semantic model, the functional data model, because it includes a programming language DAPLEX (Shipman, 1981), which is suitable for dealing with objects and is computationally rich, although incomplete. This book is based on our experiences building large systems using this language, during which it has been adapted into a more object-oriented style.

In the course of this research we have seen the value of semantic data modeling ideas. These ideas (which are described in Chapter 1) include the use of entities and relationships, the possibility of computed relationships, the use of subtypes of entities, and the importance of capturing constraints on the data such as referential integrity, single-valuedness of particular relationships, coverage of subtypes, and so on. These ideas are important because they emphasize that objects preserved over a long time must satisfy strong constraints both on their types and on their relationships to a changing network of other objects. In the absence of such constraints the whole database may become slowly corrupted and unreliable.

Many of these ideas are known to people through the much simpler entity-relationship model (Chen, 1976). It is now general practice to use some such semantic model as a prelude to designing a relational database schema, which shows that Codd's relational model is seriously lacking in these modeling concepts. This was pointed out by Smith and Smith (1977) in a classic paper on database abstractions. The continuing interest in semantic modeling techniques gave rise to a number of concrete proposals, including that of Shipman (1981), which introduced the language DAPLEX (described in Chapter 2), which was subsequently implemented both in the USA (Smith *et al.*, 1981) and in Scotland (Atkinson and Kulkarni, 1984).

The interesting thing about DAPLEX is that it combines the principles of semantic modeling with a computationally powerful programming language based on functions. Also, its comparatively simple type structure makes it easier to implement an efficient and maintainable object store for DAPLEX than for more complex models. Furthermore, it is possible to rewrite and optimize complicated DAPLEX queries on networks of objects, in much the same way that SQL queries (and updates) can be optimized for a relational database. Given that the relational model has been sold partly on the strength of SQL as a universal query language which saves the programmer from having to plan precise navigation paths and algorithms for efficient retrieval, it is clearly important to provide similar features for object-oriented databases. Chapter 6 shows that this is possible, based on experience in building such a database (Paton and Gray, 1990).

However, this book is not solely about the semantic data model viewpoint. The one area in which DAPLEX is weak is in computational completeness, thus requiring either callouts or embedding in host languages. This is true of most query languages that are currently available. Unfortunately, the embedding approach has many disadvantages. Chapter 3 reviews the research on database programming languages (DBPLs) that attempt to provide, as an alternative, a computationally complete language for data manipulation.

Chapter 3 starts with the persistent programming language PS-algol, which was used to implement the system described in Chapter 2, and which is still in use for implementing other prototype systems. PS-algol has the computational power of a programming language such as C++, but with strong type checking at compile time, and the ability to make the state of any variable or data structure persist on

disk. It is an example of a database programming language with facilities for dealing with objects on disk. Complex transactions can be written in PS-algol, as they can in other DBPLs, but it is hard to perform global optimization on the database access strategy of an arbitrary piece of PS-algol. Thus PS-algol seems more likely to serve the role of an implementation language for more high-level languages conveying the semantics of state change, for example those evolved during the TAXIS project (Mylopoulos *et al.*, 1980). In consequence, we look briefly at the lessons to be learnt from implementing EFDM in PS-algol. We conclude by discussing the more general issues of implementing semantic data models with large data volumes, especially the problems of large object stores and transaction processing.

In Chapter 4 we take a broad look at object-oriented technology, focusing upon basic object-oriented concepts, and the way in which these have evolved in both the AI field and the programming language field. We then look at how these concepts are supported in the new object-oriented database systems, both the early commercial systems and those evolving from research projects. The review of these systems examines their semantic modeling facilities, as well as tracing their ancestry back to work on DBPLs and object-oriented programming languages. Thus we begin to appreciate the power of the object-oriented paradigm, including the crucial ideas of encapsulation and inheritance. However, we question the utility of those systems that still do not provide adequate conceptual data modeling facilities, together with the means to enforce semantic constraints.

In Chapter 5 we take a more speculative look into the future as regards the storage of procedures in object-oriented databases, and their potential for software re-use. This is a very interesting and significant feature for the future development of large systems. In the process we look at some of the design decisions that were taken in modifying the functional data model so that it integrates nicely with the object-oriented database described in Chapter 6. In particular, we consider the various kinds of metadata that must be stored with functions, procedures and entities, and some ways of partitioning procedures between modules. Thus far, we have mainly seen architectures based on saved states of linked pieces of persistent code, all accessing a database of objects, but this will not suffice for the future. Future architectures for large AI frame-based systems will require objects with attached procedures, both to implement integrity constraints and to provide some form of view or behavior. Future architectures for large object-oriented systems will require procedures to be attached to class descriptors and inherited from a variety of superclasses, with specialization of behavior

The P/FDM system (Gray *et al.*, 1988) described in Chapter 6 shows one way of storing and inheriting procedures in an OODB, combined with semantic data model facilities. It is particularly interesting in its use of a Prolog query optimizer, and the fact that it is in use for a large scientific application, involving the design and analysis of large protein molecules. Thus the objects stored in the database include both entities representing parts of actual proteins and entities representing design

choices. Thus we see how an object-oriented database can function very well as a designer's database, and how it is much more suitable for this than current relational databases.

An alternative architecture for an OODB is discussed in Chapter 7 where the ADAM language (Paton, 1989) and data model are presented. Like P/FDM, its implementation adapts AI programming language technology, in its use of Prolog with object identifiers and unification. However, it is more strongly based on object-oriented concepts than P/FDM, since it emphasizes the importance of strong encapsulation and multiple inheritance. In particular, it can inherit procedure definitions from metaclasses and *mixins*. These provide a useful way of specializing behavior but without modifying the basic system code. ADAM uses these techniques to build extensions to itself which can both store and manipulate abstract descriptions of relationships, as well as other semantic modeling constructs. Furthermore, one can easily extend ADAM with methods, stored in metaclasses, that enforce semantic constraints (Diaz and Gray, 1990). This brings us back to our central theme of semantic data modeling.

The field is new and we do not claim to provide the definitive text – nor should anyone else just yet! Our main purpose is to convey experience gained from extensive experimental work, and ideas refined by discussion at many conferences and workshops. We wish to correct the balance in the literature on OODB as it currently exists, by re-emphasizing the importance of semantic data model principles, and by showing how they can be used in working object-oriented database systems. Zdonik and Maier (1990) start their collection of papers on OODBs by saying that "The fields of programming languages, artificial intelligence, and software engineering have all contributed to the use of object-oriented technology in the database area. The challenge from the database side is to integrate these threads into a single system design that maintains desired features from each field." In this book we have woven these threads into a particular pattern which we hope will both satisfy and illuminate the reader.

ACKNOWLEDGMENTS

The authors would like to express their gratitude to the many friends and colleagues who have in many different ways helped to refine the concepts discussed in this book. Any list of reasonable length is certain to be incomplete, but those whose contributions have most bearing upon what has come to be written include Malcolm Atkinson, Colin Campbell, Oscar Diaz, John Fothergill, Zhuoan Jiao, Graham Kemp and Dave Moffat.

We are also grateful to those people associated with our respective employers – Aberdeen University, Digital Equipment Corporation and Heriot-Watt University – who have either worked with us on issues related to the contents of this book,

or who have helped to provide supportive working environments. The UK Science and Engineering Research Council provided financial support for much of the work described in Chapters 5 and 6.

Figure 6.1 previously appeared in 'From Cells to Atoms', by A.R. Rees and M.J.E. Sternberg, Blackwell Scientific Publications, 1984. We are also indebted to Graham Kemp who contributed figure 6.4, the protein modeling schema shown in Section 6.6.2, and the P/FDM Daplex grammar in Appendix 1.

Contents

1

Overview of semantic data modeling

This chapter establishes the broad context in which the work reported in this book fits, namely *conceptual data modeling*. Conceptual data modeling aims to capture descriptions of objects and their behavior in the real world and to find structured representations for them in the database. Thus it is concerned with the need to understand the data and to visualize the information that it represents. It is crucial to the effective use of database technology. Conceptual data modeling has been an active area of database research from early 1960s, leading to a number of different data model proposals. These efforts have resulted in a clearer understanding of the information modeling process, captured in a concepts document from the International Standards Organization (ISO) (van Griethuysen, 1982).

We start this chapter with some preliminary definitions and the research leading to the three schema architecture proposals by ANSI/SPARC (American National Standards Institute, Standards Planning and Requirements Committee) study group. We then provide a brief discussion of the data models underlying contemporary database management systems. Readers may find both the definitions and the descriptions of classical data models rather terse. Most database textbooks cover this material in much more depth than is found here (Korth and Silberschatz, 1986; Ullman, 1988; Elmasri and Navathe, 1989; Date 1990).

The rest of the chapter starts with the discussion of why classical data models are inadequate for conceptual data modeling. We then introduce the notion of semantic data modeling followed by a discussion of the features commonly associated with semantic data models. We conclude the chapter with a brief survey of selected semantic data model proposals. Similar discussion of semantic data model concepts and a survey of selected data model proposals can be found in survey articles by

Hull and King (1987), Peckham and Maryanski (1988), and books by Tsichritzis and Lochovsky (1982), and Brodie, Mylopoulos and Schmidt (1984).

1.1 PRELIMINARY DEFINITIONS

An *information system* is a means of supplying the information needed by an organization. An information system receives the information, stores it, processes it, and provides access to it at the request of the users. When information is to be stored and processed, it needs to be coded into some descriptive form. Such coded information is called *data*. A collection of data stored on a physical media is termed a *database*. A *database system* is an information system involving four major components: database, hardware, software, and users. Users interact with the data in the database through a number of *user interfaces*.

Data as stored in a database have a certain *physical organization* on physical storage media and a certain *logical organization* as seen at the user interface. It is important to insulate the users from the physical aspects so that they are not distracted by the details of physical storage and are not inconvenienced if it is changed. A *database management system* (DBMS) is a general-purpose tool that accommodates the logical structuring, physical storage, and control of data, and provides a number of user interfaces.

An *application system* or an *application* is a part of the database system that generates the information required to serve a specific component of an organization, e.g., accounting. A *view* is a part of the database as seen by a processing activity of the application system. For example, specific activities concerning accounts payable, accounts receivable, etc., may each interact with a specific part of the database.

Databases are primarily concerned with *structured* or *formatted* data, i.e., many *instances* of data possess sufficient similarity to classify them into a *class* or *category*. This makes it possible to separate the description of the data from the actual data. The rules that the instances of a class are expected to obey are specified once, in a *schema*. Hence, the schema contains the description of the data.

A *data model* is the primary tool for designing a database. The basic components of such a data model include a set of rules to describe the structure and meaning of data in a database and the atomic operations that may be performed on the data in that database. Thus, a data model **M** can be defined as consisting of two parts: a set of generating rules, **G**, and a set of operations, **O** (Tsichritzis and Lochovsky, 1982). **G** defines the allowable *structures* for the data as a set of schemas **S**. The set of generating rules **G** can be partitioned into two parts: the structure specification **Gs** and the rule specification **Gr**. The generators **Gs** generate the categories and structures of a schema and the generators **Gr** generate the inferences and the constraints associated with a schema. A schema **S** therefore consists of two